MISB Motion Imagery Standards Board **STANDARD** **Floating Point to Integer Mapping**	**MISB ST 1201.4** **28 February 2019**

1 Scope

This standard describes the method for mapping floating-point values to integer values and the reverse, mapping integer values back to their original floating-point value to within an acceptable precision. There are many ways of optimizing the transmission of floating-point values from one system to another; the purpose of this standard is to provide a single method for use by all MISB metadata standards. This standard supports all floating-point ranges and valid precisions. This standard provides a method for a forward and reverse linear mapping of a specified range of floating-point values to a specified integer range of values based on the number of bytes desired for the integer value. Additionally, it provides a set of special values to transmit non-numerical "signals" to a receiving system. This standard is dependent on context from an invoking standard (or another document), called a Parent Document.

2 References

[1] IEEE 754-2008 Standard for Floating-Point Arithmetic [and Floating-Point formats], 2008.
[2] MISB ST 0601.15 UAS Datalink Local Set, Feb 2019.

3 Acronyms

IEEE	Institute of Electrical and Electronics Engineers
IMAPA	[Floating Point to] Integer Mapping using Starting Point A
IMAPB	[Floating Point to] Integer Mapping using Starting Point B
KLV	Key Length Value
MISB	Motion Imagery Standards Board
msb	Most Significant Bit
ST	Standard

4 Revision History

Revision	Date	Summary of Changes
ST 1201.4	02/28/2019	• Fixed issue with IMAPA Lbits computation • Deprecated Req's -01 through -08, -10 & -11 as they are non-testable; recast as algorithm principles

		• Edits to text to improve readability • Updated reference [2]

5 Terms and Definitions

ceiling

Defined as rounding any non-integer value up toward +∞. The notation used is $\lceil x \rceil$ and the definition of the ceiling function is: $\lceil x \rceil = \min\{n \in \mathbb{Z} | n \geq x\}$. Examples: $\lceil -1.1 \rceil = -1$; $\lceil 1.1 \rceil = 2$. Note: Microsoft Excel (pre-2010) does not perform this operation correctly.

floor

Defined as rounding any non-integer value down towards -∞. The notation used is $\lfloor x \rfloor$ and the definition of the floor function is: $\lfloor x \rfloor = \max\{m \in \mathbb{Z} | m \leq x\}$. Examples: $\lfloor -1.1 \rfloor = -2$; $\lfloor 1.1 \rfloor = 1$. Note: Microsoft Excel (pre-2010) does not perform this operation correctly.

truncate

A function that removes the fractional part of a real number (e.g., **truncate** (100.2) = 100).

round

A function has different modes but for this standard round means $\lfloor x + 0.5 \rfloor$ if x≥0 and $\lceil x - 0.5 \rceil$ if x<0.

precision (From [1])

The maximum number, p, of significant digits in a numerical format, or the number of digits a result is rounded to. Note: This is not the same definition that refers to repeatability of measurements as used in other MISB documents and references.

6 Introduction

Systems transmit and receive measured and computed floating-point values between systems. Oftentimes, the values do not fully utilize the floating-point range or precision afforded, and thus provide an opportunity to reduce the number of bytes representing the data. Additionally, systems need to communicate non-numeric special values or "signals"; for example, a sensor value indicating "beyond measurement range" or in cases where a divide by zero occurs (i.e., +infinity). This standard applies to IEEE 754 [1] floating-point values represented in 16, 32, 64, and 128-bit precision, and includes IEEE special values for infinity, and NaN's (Not-a-Number).

For convenience, Section 8 succinctly states the algorithm for developers to implement and Section 9 informatively shows the derivation of the algorithm and format.

7 Algorithm Principles

The following principles underlie the IMAP algorithm:

- The binary representation of a floating-point value is in the format of an IEEE 754-2008 floating-point value.

- The algorithm performs the mapping based on a [user] specified floating-point range from a min value to a max value – inclusive (including the end points).
- The algorithm is a linear mapping from a floating-point value to a non-negative integer value of [user] defined length in bytes.
- The binary representation of a non-negative integer value is in the format of a standard unsigned integer, with the length, in bytes, defined by the [user's] implementing Parent Document.
- The algorithm provides an inverse mapping from a non-negative integer value to the original floating-point value within a [user] specified precision. (Note: the mapping algorithm can supply better precision than what the user specifies. The defined length and precision are related – given one the other can be computed but both cannot be specified by the user).
- If the [users] floating-point range includes 0.0, then the algorithm maps 0.0 exactly to a specific integer value, called a "zero offset."
- All negative floating-point values map to an integer that is less that the zero offset.
- All positive floating-point values map to values greater than or equal to the zero offset.
- The resulting format provides a set of special values which map to the IEEE 754-2008 special values, including: ± infinity, ±QNan and ±SNan (note: negative zero is not included in this requirement).
- The resulting format provides a bit space for custom signals to be sent by the user as defined by the [users] implementing Parent Document (e.g., MISB ST 0601).
- The algorithm is designed for fewest operations as possible (during the actual mapping) for computational efficiency.

8 Mapping Algorithm and Integer Format

This section describes the algorithm for mapping the floating-point values to integer values, and the reverse mapping of integers back to floating-point values. Additionally, this section defines the notation to use in any MISB Parent Document when "invoking" this standard. Section 9 is an informative section which describes the development of this algorithm in detail. The algorithm is based on two steps as shown in Figure 1:

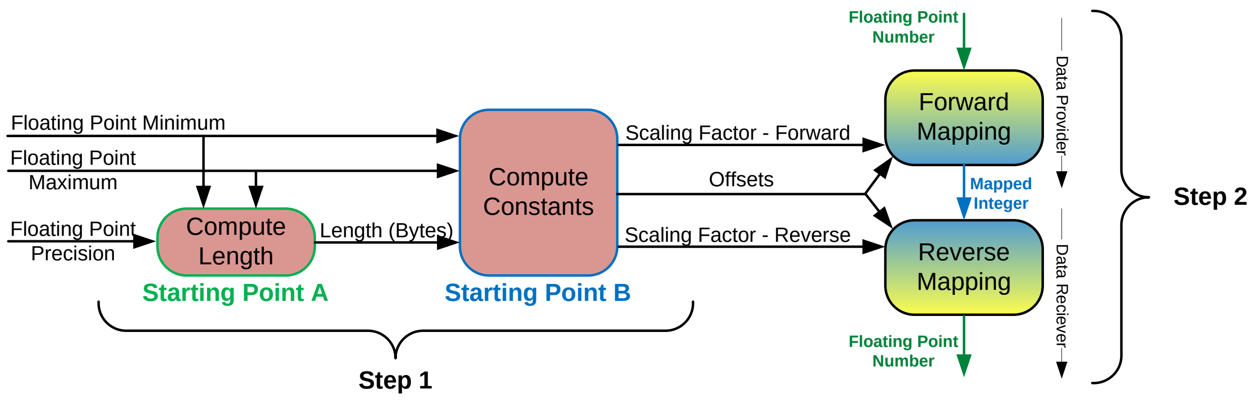

Figure 1: Functional view of the two-step conversion process

Step 1: The first step computes one-time use constants, and the second step is the forward or reverse mapping. There are two “starting points” for the first step. With known precision, minimum, and maximum values Starting Point A computes the mapped value length for use in Starting Point B. With known length, minimum, and maximum values Starting Point B computes the Scaling Factor Constants for the forward and reverse mapping step (Step 2). Starting Point A equates to IMAPA and Starting Point B equates to IMAPB as discussed in Section 8.3.

Step 2: The second step uses the constants computed from Step 1 to perform floating-point-to-integer and/or integer-to-floating-point mappings – provided the floating-point minimum, maximum, and length do not change. Figure 1 illustrates the interaction of the two steps.

Steps 1 and 2, described in the sections below, use the standard order of operations (i.e., operations in parenthesis first, exponents/roots second, multiplication/division third, addition/subtraction fourth).

8.1 Step 1: Scaling Factors & Offsets

Step 1 has two starting points based on whether the floating-point precision is known or the length (in bytes) of the resulting mapped integer is known.

8.1.1 Starting Point A: Specifying Precision

Input
• a = Floating-Point Minimum, range is from smallest float value to largest float value • b = Floating-Point Maximum, range is from smallest float value to largest float value • g = Floating-Point Precision, range is from zero to largest float value • Note: a<b and g<b-a
Algorithm **Compute Length L (Bytes)**
1. Lbits = **ceiling**(log$_2$(b-a))-**floor**(log$_2$(g)) + 1 (note: +1 is for special values, see below) 2. L = **ceiling**(Lbits/8) 3. Follow steps in Starting Point B

8.1.2 Starting Point B: Specifying Length

Input
• a = Floating-Point Minimum, range is from smallest float value to largest float value • b = Floating-Point Maximum, range is from smallest float value to largest float value • L = Length of resulting mapped integer (note: this includes an extra bit for the special values, see below) • Note: a<b
Algorithm **Compute constants for forward and reverse mapping**
1. bPow = **ceiling**(log$_2$(b-a)) 2. dPow = 8*L-1

3. sF = 2^(dPow -bPow)
4. sR = 2^(bPow- dPow)
5. Z_{offset} = 0.0
6. if (a<0 and b>0) then Z_{offset} = sF*a-**floor**(sF*a)

8.2 *Step 2: Forward and Reverse Mapping*

Step 1 computes the scaling factors for the forward and reverse mapping in Step 2. These operations are computationally optimal, so the mapping requires only one multiply and two adds. Because the multiplication uses power of two values, an option is to implement the multiplication using floating point binary shifts (i.e., in hardware constrained environments).

8.2.1 Forward Mapping

Input

- sF = Forward scaling factor (computed in Step 1)
- a = Floating-Point Minimum
- Z_{offset} = Zero-point offset (computed in Step 1)
- x = floating-point value to map to integer y

Algorithm
Compute mapped integer value y

1. If x is a special value, then:
   y = Table Lookup to map to specified bit pattern (see Section 8.2.3 below)
2. Else x is a normal floating-point number then:
   a. y=**truncate**(sF*(x-a)+Z_{offset})
      i. Note: Since sF is a power of 2, an option is to use a floating-point shift instead of a multiplication
      ii. Note: Convert y into L number of bytes values (i.e., the lower L number of bytes is the "value" to send)

Developer Notes

1) IEEE 754 floating-point numbers do not explicitly represent the most significant 1-bit in the mantissa, so if performing a manual shift (i.e., the CPU does not support a floating-point shift), prepend a leading bit to the mantissa before shifting and converting to an integer. As a simple example, the IEEE 754 mantissa for 6 is 1000..., it first becomes 11000..., then ...00000110.
2) If special values are not in use for a value, an optimization of the forward mapping is to skip the "special values check" and omit the table look-up. Preserve the extra bit (special value bit) in the data format but remove the conditional check (Step 1 above).

8.2.2 Reverse Mapping

Input

- sR = Reverse scaling factor (computed in Step 1)
- a = Floating-Point Minimum
- Z_{offset} = Zero-point offset (computed in Step 1)
- y = integer value to map to floating-point value x

Algorithm
Compute mapped floating-point value x

```
Define: Bit(q,y) = the qth bit of y. msb = Most Significant Bit
    1. special_value = Bit(msb,y) & Bit(msb-1,y)
    2. If special_value equals 1 then
        a. x=table lookup to map to specified floating-point value (Section see 8.2.3 Table 2
           below)
    3. Else y is a normal value then
        a. $x=sR*(y-Z_{offset}) + a$
            i. Note: Since sR is a power of 2, an option is to use a floating-point shift instead
               of a multiplication
```

Developer Notes

1) IEEE 754 floating-point numbers do not explicitly represent the most significant 1-bit in the mantissa, so if performing a manual shift (i.e., the CPU does not support a floating-point shift), prepend a leading bit to the mantissa before shifting and converting to an integer. As a simple example, the mantissa for 6 is 1000..., it first becomes 11000..., then ...00000110.
2) Before applying this algorithm, verify the length is the correct length for the pre-computed sR value, see Section 8.4 for further information.

8.2.3 Special Value Mappings

In addition to normal numeric values this standard defines a list of special values to indicate unusual circumstances, such as a gimbal lock, division by zero, etc. This standard defines existing standardized values (based on IEEE-754) and a set of MISB definable values.

The first two bits indicate either a normal or special value as shown in Table 1. The msb is zero for all normal mapped values except for the maximum floating-point value, b, but only if (b-a), as specified by the user, is a power of 2.

Table 1: Bit Patterns

Bits			Description
b_n (msb)	b_{n-1}	$b_{n-2} – b_0$	
0	x	x	Normal mapped value (x=1 or 0)
1	0	0	Normal mapped value. This is the max value of the normal mapping values; this is the only normal value with msb=1.
1	0	x	(At least one x bit = 1) – Reserved section of bit space
1	1	x	Special value indicator (x=1 or 0) – see Table 2

If both the first and second msb are set to one, then the third, fourth and fifth msbs indicate which special value it is, as shown in Table 2. Since the mapped integer has fewer bytes than the original floating-point number not all the source NaN Identifiers can be defined. The default NaN Identifier is a value with all zeros.

Table 2: Special Value Bit Patterns

Name	Most Significant Bits					Other bits	Description - IEEE-754 values (or user defined)
	b_n (msb)	b_{n-1} (Special)	b_{n-2} (Sign)	b_{n-3} (NaN)	b_{n-4}	$b_{n-5} – b_0$	
+Infinity	1	1	0	0	1	Zero Filled	Positive Infinity (+∞)
-Infinity	1	1	1	0	1	Zero Filled	Negative Infinity (-∞)
+QNaN	1	1	0	1	0	NaN Id*	Positive Quiet NaN (Not a Number)
-QNaN	1	1	1	1	0	NaN Id*	Negative Quiet NaN
+SNaN	1	1	0	1	1	NaN Id*	Positive Signal NaN – “Other bits” are the signal value
-SNaN	1	1	1	1	1	NaN Id*	Negative Signal NaN – “Other bits” are the signal value
Reserved	1	1	1	0	0	Reserved	Reserved
User Defined	1	1	0	0	0	User Defined	“Other bits” enumerate user defined signals
Reserved	1	0	1	X	X	Reserved	Reserved

*This standard defines only one NaN Identifier, i.e., fill bits $b_{n-5} – b_0$ with zeros. The MISB will define NaN Identifier’s as necessary; please contact the MISB to define new values. The values need to be acceptable universally and not special values for a specific processor, application or program.

This standard does not define a special value for the IEEE 754 negative zero value; use positive zero to represent negative zero values.

Requirement	
ST 1201.1-12	All negative zero values shall be mapped to a positive zero and sent as a normal (non-special) value.

In addition to the IEEE bit values a set of MISB-defined bit patterns are available for use within a Parent Document. For example, in MISB ST 0601 [2] the sensor depression angle value could include a bit pattern to indicate gimbal lock; this bit pattern would then only be valid for that value in ST 0601.

8.3 MISP and MISB Document Notation

When using or "invoking" this standard in a MISB Parent Document it is important to be clear as to the starting point, and the values used as "inputs."

Requirement(s)	
ST 1201.1-13	When the float precision is fixed within the implementing standard the notation for Starting Point A shall be IMAPA (<min float>, <max float>, <float precision>).
ST 1201.2-16	When the float precision is computed or defined at runtime the notation for Starting Point A shall be IMAPA (<min float>, <max float>).
ST 1201.1-14	When the length is fixed within the implementing standard the notation for Starting Point B shall be IMAPB (<min float>, <max float>, <length bytes>).
ST 1201.2-17	When the length is computed or defined at runtime the notation for Starting Point B shall be IMAPB (<min float>, <max float>).
ST 1201.1-15	When adding user defined bit patterns, they shall be listed immediately after the IMAPA or IMAPB notation.

Example: Starting Point A:

- IMAPA(-200.0, 3000.0, 0.5)
- Interpretation: map a value in the range from -200.0 to 3000.0 using, at the minimum, increments of 0.5.

Example: Starting Point A (with computed precision):

- IMAPA(-200.0, 3000.0)
- Interpretation: map a value in the range from -200.0 to 3000.0 using a precision computed at runtime.

Example: Starting Point B:

- IMAPB(-200.0, 3.000, 3)
- Interpretation: map a value in the range from -200.0 to 3000.0 using 3 bytes (see Section 8.4 for comments on length).

Example: Starting Point B (with computed length):

- IMAPB(-200.0, 3.000)
- Interpretation: map a value in the range from -200.0 to 3000.0 using a length computed at runtime (see Section 8.4 for comments on length).

8.4 Length Processing

Lengths computed or provided when defining the mapping (IMAPA, IMAPB) are the recommended number of bytes to use. Depending on the form used to transmit KLV data (Universal Sets, Local Sets, Individual Items), it is possible to use a different [KLV] length (L) instead of the recommended or computed [IMAP] length. When using a different length, it is important to compute the constants needed to do the forward and reverse mapping based on the KLV supplied length.

9 Algorithm Development – Informative

The development of the algorithm for this standard is discussed in the following sections:

1) Goal
2) Mapping and Inverse Mapping
3) Error Analysis
4) Simplification
5) Zero Matching
6) Computing d
7) Power of 2 adjustments for b
8) Computing L from Precision
9) Summary

9.1 Goal

The goal is to map a floating-point range (F_R) to an integer range (I_R) with a chosen maximum precision (g) using the smallest integer range (I_R) and the least amount of computation. In addition, when a F_R range includes zero the algorithm must map zero to a specific integer value (zero offset), and all negative floating-point values must map to values less than the zero offset. Refer to Section 7 for the requirements for this algorithm.

Examples:

1) Map floating-point range 0.1 to 3.1 to 2 bytes
2) Map -1.0 to +1.0 to 2 bytes
3) Map -0.5 to -0.3 to 1 byte
4) Map -3.14159265 to 3.14159265 to 4 bytes
5) Map floating-point range 0.1 to 3.1 with precision of 0.01 or better
6) Map -1.0 to +1.0 to 2 bytes with precision of 0.001 or better
7) Map -0.5 to -0.3 to 1 byte with precision of 0.01 or better

8) Map -3.14159265 to 3.14159265 with precision of 0.00314159265 or better

9.2 Mapping and Inverse Mapping Definition

Given a $F_R = \{x \in \mathbb{R} | x \in [a, b]\}$ and $I_R = \{y \in \mathbb{Z} | y \in [c, d]\}$

Let A(x) be a linear mapping from F_R to I_R and $A^{-1}(y)$ be a mapping from I_R to F_R.

Figure 2 illustrates the linear mapping, where A(x) (blue dotted line) maps from F_R (x-axis) to I_R (y-axis) via the red line; $A^{-1}(y)$ (green dotted line) maps from the I_R (y-axis) to F_R (x-axis) via the red line.

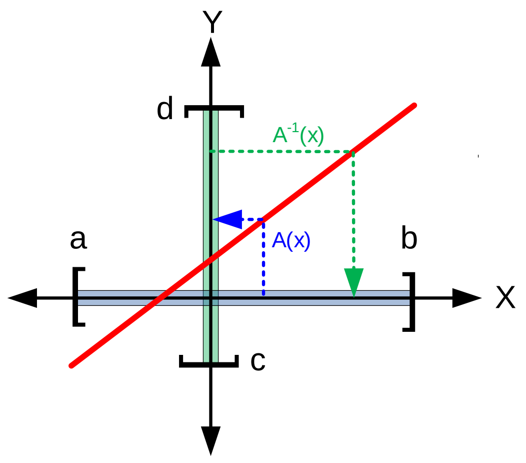

Figure 2: A(x) maps F_R to I_R; $A^{-1}(x)$ maps I_R to F_R

Using the two-point method for defining the mapping:

(Eq 1) $$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) = y - c = \frac{d-c}{b-a}(x - a)$$

After rearranging, since y is an integer, truncate or round the right-hand side of the equation before computing y:

(Eq 2) $$y = I\left(\frac{d-c}{b-a}(x - a) + c\right)$$

Function $I(\ \ )$ is either the truncation or rounding function (discussed in Section 9.3). (Eq 2) equation defines A(x), the forward mapping of any real value (floating point) range to an integer range.

(Eq 3) defines the inverse of (Eq 2) $A^{-1}(y)$:

(Eq 3) $$x = (y - c)\frac{b-a}{d-c} + a$$

The above equations are for all a, b, c, and d values; however, the following sections describe additional simplification and improvements.

9.3 Error Analysis

The mapping function is not a one-to-one function, which means the mapping and subsequent inverse mapping introduce errors. For each value y, in I_R, a unique range of x values in F_R maps to a single y value; define this range of x values as $x_{original}$. When the integer value y is inverse mapped back to an x value it will only match one of the values of $x_{original}$. Therefore, the forward mapping of an x value to an integer, followed by the reverse mapping back to a floating-point x value induces an error (except in special cases). The tolerance of this error is dependent on the application, so the implementer of this standard must ensure the use of large enough c and d values to create acceptable minimum error – more on this topic in other sections below. Figure 3 shows a subset of values, $x_{original}$ (in red), mapping to a single integer value y. This also shows the single integer value, y, maps back to only one value, x_{final} (green), in the floating-point range

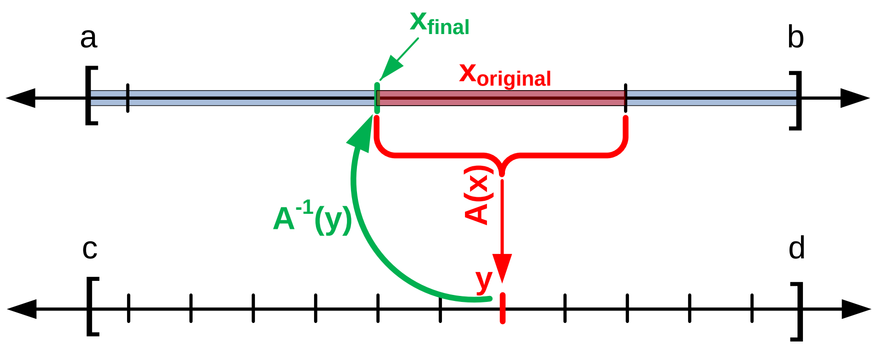

Figure 3: Many-to-one forward mapping and one-to-one reverse mapping

Example 1:

Let a=0.0, b=5.0 and c=0, d=5; then $y = I\left(\frac{5-0}{5.0-0.0}\,(x-0.0)+0\right) = I\left(\frac{5}{5}\,x\right) = I(x)$

For this example, use I(x) = **truncate**(x); therefore, all values of the $x_{original}$ range map to a single y value. The inverse mapping, maps y to a single x value, x_{final}. For example, all values in $x_{original} = \{x \in [1.0, 2.0)\}$ map to y=1, and the inverse mapping value of y=1 map to x_{final} = 1.0 for all values in the $x_{original}$ range.

The error is the absolute difference between the starting value and the result of forward and inverse mapping. For example, an x value of 1.4 maps to y=1, then inverse maps back to x=1.0, producing an error of $|1.4 - 1.0| = 0.4$.

To compute the amount of error: $E(x) = |x - A^{-1}(A(x))\,|$. Graphing the error for Example 1 produces the saw-tooth graph shown in Figure 4.

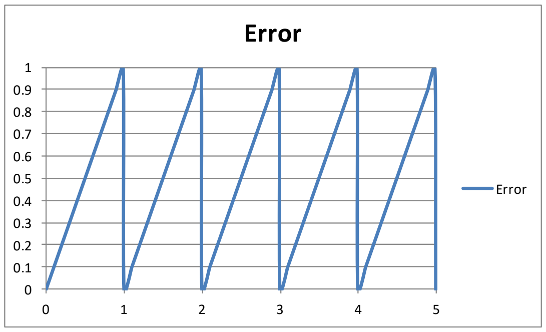

Figure 4: Error in forward mapping of Example 1

The maximum error in Example 1 is slightly less than 1.0; thus, implementors may use this mapping function if this level of error is acceptable. If less error is desired, it is a simple matter to use a larger integer range; for example, doubling the range cuts the error in half.

There are multiple ways of converting a floating-point number to an integer, including "normal" rounding, truncating, round away from zero, round to zero, etc. Normal rounding rounds down when the fractional value is less than 0.5 and rounds up when the fractional value is 0.5 or greater. Considering error, the best conversion technique is normal rounding shown in Figure 5.

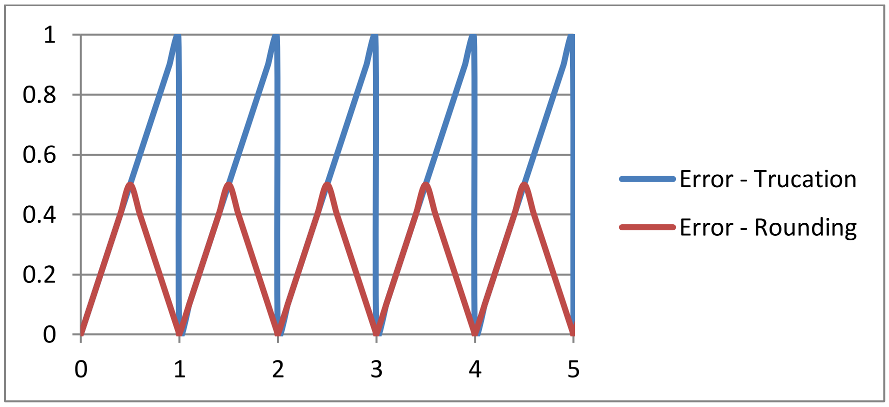

Figure 5: Rounding versus Truncation error

<u>However, to support the zero matching requirements a truncation must be used instead</u> (see Section 9.5), so (Eq 2) is changed to include truncation. Since x ranges from [a, b], all values of (x-a) will be greater than or equal to zero so **truncate** and **floor** perform the same function. (Eq 2) uses the **floor** notation $\lfloor \quad \rfloor$ throughout the remainder of this section:

$$\text{(Eq 4)} \qquad y = \left\lfloor \frac{d-c}{b-a}(x-a) + c \right\rfloor$$

9.4 Simplifications

There are two simplifications to (Eq 4) to reduce overall (floating point) errors and enable some optimizations: shifting the integer range and shifting the floating-point range.

The integer range (I_R) values, c and d, are arbitrary for this algorithm, so in this standard the minimum integer value is forced to always equal zero. When c is zero, d is the maximum value desired by the implementer for the desired precision. An additional benefit with the minimum value equal to zero is the resulting value is never negative, therefore it is an unsigned integer requiring no sign bit (nor 2's complement formatting). Setting c=0 results in the following change to (Eq 4):

(Eq 5)	$y = \left\lfloor \frac{d}{b-a} (x-a) \right\rfloor$

Using substitution, letting x'=x-a, shifts the floating-point range, F_R. Shifting the range F_R by a constant (a) produces a new range called F'_R. $F'_R = \{x' \in \mathbb{R} | x' \in [0, b']\}$ where a'=a-a=0 and b' = b-a. With this change the minimum value of F'_R is zero, so (Eq 5) and (Eq 3) are respectively simplified to:

(Eq 6)	$y = \left\lfloor \frac{d}{b'} x' \right\rfloor = \left\lfloor \frac{d}{b'} (x - \mathrm{a}) \right\rfloor$
(Eq 7)	$x = \frac{b'}{d} y + a$

9.5 Zero Matching

When the floating-point range F'_R includes zero it is desirable to have the zero value directly map to an integer in I_R. This zero-integer value is the "Zero Point" or I_{zero}. Furthermore, all negative values in F_R map to integers less than I_{zero} and all positive values map to integers equal to or greater than I_{zero}.

Example 2:

With F_R = -0.9 to 1.1 and d=11, Table 3 lists example x values in column 1. Column 2 is the x value adjusted to the new F'_R range, x'. Column 3 is the mapping before converting to an integer. Columns 4 through 7 show different integer conversions of the mapping. Note: g=2.0/11 = 0.1818.

Table 3: Various mappings for Example 2

1	2	3	4	5	6	7
X	x'	(d/b')x'	**ceiling**	**floor**	**truncate**	**round**
-0.9	0	0	0	0	0	0
-0.71818	0.181818	1	1	1	1	1
-0.53636	0.363636	2	2	2	2	2
-0.35455	0.545455	3	3	3	3	3
-0.17273	0.727273	4	4	4	4	4
-0.08182	0.818182	4.5	5	4	4	5
0	0.9	4.95	5	4	4	5
0.009091	0.909091	5	5	5	5	5
0.1	1	5.5	6	5	5	6
0.190909	1.090909	6	6	6	6	6
0.372727	1.272727	7	7	7	7	7
0.554545	1.454545	8	8	8	8	8
0.736364	1.636364	9	9	9	9	9
0.918182	1.818182	10	10	10	10	10
1.1	2	11	11	11	11	11

To illustrate what happens when not using a Zero Point, the red row in Table 3 shows the original zero (0) value (in column 1) and the value equal to 0.9 (in column 2) after shifting the floating-point range. The 0.9 value maps to 4.95 (column 3) before conversion to an integer. Based on the integer conversion technique (columns 4-7), the original zero value maps to a value of 4 or 5, which is the same value as a preceding negative value (in yellow) of -0.08182. When mapping the integer value back to a floating-point value the original value of zero (in column 1) becomes -0.08182; a zero value becomes negative after the mapping. This behavior is undesirable; instead, the zero in F_R should map to an integer that maps back exactly to zero in F_R.

To achieve this, the zero in F_R must map exactly to a whole integer (i.e., before converting to an integer) – this integer value is the Zero Point, I_{zero}. With the Zero Point, all negative values in F_R map to integers less than I_{zero}. This means all negative values forward and reverse map back to negative values; the zero value forward and reverse maps back to zero; and all positive values forward and reverse map back to positive values or zero (if they are close to zero).

To implement the Zero Point the whole mapping equation is shifted, so the zero in F_R becomes an integer value (I_{zero}); an offset is added before the integer conversion. Furthermore, truncation (or floor) ensures all negative values are below I_{zero}; conversely, ceiling and rounding will force negative values to be identical to I_{zero}. The Zero Point shift is different than changing the bounds (a,b or c,d) of the equations; the slope of the linear mapping is not affected by this change.

The adjustment to the mapping equations (Eq 5) and (Eq 6) are:

(Eq 8) $\mathrm{Z_{offset}} = \frac{d}{b'}a - \left\lfloor \frac{d}{b'}a \right\rfloor$, Note: $0.0 \le \mathrm{Z_{offset}} < 1.0$

(Eq 9) $y = \left\lfloor \frac{d}{b'}(\mathrm{x} - \mathrm{a}) + Z_{\mathrm{offset}} \right\rfloor$

(Eq 10) $x = \frac{b'}{d}(y - Z_{\mathrm{offset}}) + \mathrm{a}$

Re-computing Example 2 with these changes results in:

$$\mathrm{Z_{offset}} = \frac{11}{2.0}(-0.9) - \left\lfloor \frac{11}{2.0}(-0.9) \right\rfloor = -4.95 - (-5.0) = 0.05$$

Table 4 shows the mapping with the $\mathrm{Z_{offset}}$ (0.05) that correlates to the Zero Point ($\mathrm{I_{zero}}$), value of 5. All negative x values map to 4 or less and all positive values (and zero) map to 5 or greater. Do not use ceiling and rounding; as shown in column 4 and 7 they map positive and zero values with negative value.

Table 4: Example 2 recomputed using $\mathrm{Z_{offset}}$

1	2	3	**4**	**5**	**6**	**7**
x	x'	(d/b')x' + $\mathrm{Z_{offset}}$	**ceiling**	**floor**	**truncate**	**round**
-0.9	0	0.05	1	0	0	0
-0.71818	0.181818	1.05	2	1	1	1
-0.53636	0.363636	2.05	3	2	2	2
-0.35455	0.545455	3.05	4	3	3	3
-0.17273	0.727273	4.05	5	4	4	4
-0.08182	0.818182	4.55	5	4	4	5
0	0.9	5	5	5	5	5
0.009091	0.909091	5.05	6	5	5	5
0.1	1	5.55	6	5	5	6
0.190909	1.090909	6.05	7	6	6	6
0.372727	1.272727	7.05	8	7	7	7
0.554545	1.454545	8.05	9	8	8	8
0.736364	1.636364	9.05	10	9	9	9
0.918182	1.818182	10.05	11	10	10	10
1.1	2	11.05	12	11	11	11

9.6 Computing d

In all the previous equations, the number of bytes the user specifies with value L determines the value d. The value d uses all the bits in the bytes including the most significant bit (msb); however, there is only one case which uses the msb for a mapped value. When the user specifies a maximum value, b, which is a power of 2, the mapped integer value can include the msb. In this case the only time the msb is used is when the floating-point value to map is the maximum value. When this happens, the msb is set to 1 and all other bits are set to zero. All other uses of the msb signal special values, see Section 8.2.3.

(Eq 11) $\mathrm{N}_{\text{bits}} = 8 * L - 1$

(Eq 12) $d = 2^{\mathrm{N}_{\text{bits}}}$

(Eq 12) shows d is a power of two, which affords an optimization capability described in Section 9.7.

9.7 Power of 2 Adjustment for b

To potentially improve algorithm efficiency and reduce floating-point rounding errors, this standard adjusts the floating-point range, F_R, to be a power of two. The cost of this adjustment is a slight reduction in the precision.

Examining the equation for the mapping shows the mapping is primarily a simple division, multiplication, and addition: $y = \left\lfloor \frac{d}{b'}(x - a) + Z_{\text{offset}} \right\rfloor$. Floating-point division and multiplication can easily introduce small numeric floating-point rounding errors. To reduce this error and to provide a more efficient algorithm b' is adjusted to be based on a power of two.

(Eq 13) $\bar{b} = 2^{\lceil log_2 b' \rceil}$

With the adjustments to b' (Eq 9) and (Eq 10) are updated as follows:

(Eq 14) $y = \left\lfloor \frac{d}{\bar{b}}(\mathrm{x} - \mathrm{a}) + Z_{\text{offset}} \right\rfloor$

(Eq 15) $x = \frac{\bar{b}}{d}\,(y - Z_{\text{offset}}) + \mathrm{a}$

Since d and $\bar{b}$ are both powers of two, an optimization is to pre-compute the forward and reverse scaling factors:

(Eq 16) $\frac{d}{\bar{b}} = S_F = 2^{(N_{bits} - \lceil log_2 b' \rceil)}$

(Eq 17) $\frac{\bar{b}}{d} = S_R = 2^{(\lceil log_2 b' \rceil - N_{bits})}$

Adjusting (Eq 14) and (Eq 15) produces the final mapping equations:

(Eq 18) $y = \lfloor \mathrm{S_F}(\mathrm{x} - \mathrm{a}) + Z_{\text{offset}} \rfloor$

(Eq 19) $x = \mathrm{S_R}\,(y - Z_{\text{offset}}) + \mathrm{a}$

9.8 Computing L from known precision

When using this standard there are two starting points based either on: (Starting Point A) the minimum, maximum and known floating-point precision, or (Starting Point B) the minimum, maximum and desired length. The preceding sections describe the steps for computing scaling factors based on Starting Point B, knowing the minimum, maximum and number of bytes (L). This section shows the computation of L based on the known minimum, maximum and known floating-point precision (Starting Point A).

Because the scaling factors (S_F and S_R) in (Eq 18) and (Eq 19) are rounded up to powers of two, the users precision value, g, is rounded down (providing more precision) to a power of two, g', to ensure the users required precision is maintained in the mapping.

(Eq 20) $g' = 2^{\lfloor \log_2 g \rfloor}$

Given a, b, and g'; where a is the minimum value of the range, b is the maximum value of the range, and g' is the floating-point precision rounded down, the number of bits, L_{bits}, needed to represent this number for the mapping is:

(Eq 21) $L_{bits} = \left\lceil log_2(\frac{b-a}{g'}) \right\rceil + 1$ Note: +1 is for special values bit.

Combining equations (Eq 20) and (Eq 21) provides a single computation for the bits in (Eq 22). (Eq 23) shows the computation of the number of bytes from L_{bits}.

(Eq 22) $L_{bits} = \lceil log_2(b-a) \rceil - \lfloor \log_2 g \rfloor + 1$

(Eq 23) $L = \left\lceil \frac{L_{bits}}{8} \right\rceil$

(Eq 11) uses the value of L.

9.9 Summary

The following shows a consolidation of all the equations needed to map values to and from floating point to integer.

Starting Point A: User specifies a, b, and g (the desired precision to use)

<u>One-time computation:</u>

$L_{bits} = \lceil log_2(b-a) \rceil - \lfloor \log_2 g \rfloor + 1$

$L = \left\lceil \frac{L_{bits}}{8} \right\rceil$

Now a, b and L are defined for starting point B.

Starting Point B: User specifies, a, b, and L (number of bytes to use)

<u>One-time computation:</u>

$b_{pow} = \lceil log_2(b-a) \rceil$

$d_{pow} = 8 * L - 1$ (Number of bits for d)

$S_F = 2^{(d_{pow} - b_{pow})}$ (Scaling for forward mapping, note same as bitwise shift)

$S_R = 2^{(b_{pow} - d_{pow})}$ (Scaling for reverse mapping, note same as bitwise shift)

$Z_{offset} = 0$ (Default offset to zero)

If (a<0 and b>0) then $Z_{offset} = S_F a - \lfloor S_F a \rfloor$ (only compute if condition met)

Once computing the above values, use S_F, S_R, and Z_{offset} for forward mapping and reverse mapping every x and y value, respectively.

Per x value:

$$y = \lfloor S_F(x - a) + Z_{\text{offset}} \rfloor$$

Per y value:

$$x = S_R(y - Z_{\text{offset}}) + a$$

See Section 8.2.3 for information on handling special values (i.e., infinity, NaN, etc.) of x and y.

10 Deprecated Requirements

The following requirements are deprecated because they are not testable.

Requirement(s)	
ST 1201.1-01 (Deprecated)	The binary representation of a floating-point value shall be in the format of an IEEE 754-2008 floating-point value.
ST 1201.1-02 (Deprecated)	The algorithm shall perform the mapping based on a [user] specified floating-point range from a min value to a max value – inclusive (including the end points).
ST 1201.1-03 (Deprecated)	The algorithm shall be a linear mapping from a floating-point value to a non-negative integer value of [user] defined length in bytes.
ST 1201.1-04 (Deprecated)	The binary representation of a non-negative integer value shall be in the format of a standard unsigned integer, with the length, in bytes, defined by the [user's] implementing Parent Document.
ST 1201.1-05 (Deprecated)	The algorithm shall provide an inverse mapping from a non-negative integer value to the original floating-point value within a [user] specified precision. (Note: the mapping algorithm can supply better precision than what the user specifies. The defined length and precision are related – given one the other can be computed but both cannot be specified by the user).
ST 1201.1-06 (Deprecated)	If the [users] floating-point range includes 0.0, then the algorithm shall map 0.0 exactly to a specific integer value, called a "zero offset."
ST 1201.1-07 (Deprecated)	All floating-point values that are negative shall map to an integer that is less that the zero offset.
ST 1201.1-08 (Deprecated)	All values that are positive shall map to values greater than or equal to the zero offset.
ST 1201.1-09 (Deprecated)	The algorithm shall be designed to be as few operations as possible (during the actual mapping) for computational efficiency.
ST 1201.1-10 (Deprecated)	The resulting format shall provide a set of special values which map to the IEEE 754-2008 special values, including: ± infinity, ±QNan and ±SNan (note: negative zero is not included in this requirement).
ST 1201.1-11 (Deprecated)	The resulting format shall provide a bit space for custom signals to be sent by the user as defined by the [users] implementing Parent Document (e.g., MISB ST 0601).

11 Appendix A – Usage Examples

Table 5 shows examples of ranges, precisions, and number of bytes along with the invoking document IMAP notation.

Table 5: Usage examples

Name	Min	Max	Precision	Number of Bytes	IMAP Statement
Altitude	-900.0	19000.0	Don't care	2	IMAPB(-900, 19000, 2)
Covariance	-1.0	1.0	Don't care	2	IMAPB(-1, 1, 2)
Radians	-3.14159265	3.14159265	Don't care	3	IMAPB(-3.14159265, 3.14159265, 3)
Altitude	-900.0	19000.0	0.5	Don't care	IMAPA(-900, 19000, 0.5)
Covariance	-1.0	1.0	.0001	Don't care	IMAPA(-1, 1, .0001)
Radians	-3.14159265	3.14159265	0.00314159265	Don't care	IMAPA(-3.14159265, 3.14159265, 0.00314159265)
Small Range	0.1	0.9	Don't care	2	IMAPB(0.1, 0.9, 2)

The following illustrates a complete example of the computations for mapping an altitude range using, IMAPA(-900, 19000, 0.5).

Example 3: Altitude (Starting point A):

- a = Floating-Point Minimum = -900.0
- b = Floating-Point Maximum = 19,000.0
- g = Floating-Point Precision = 0.5

Starting Point A - Algorithm: Compute Length L (Bytes)

1. Lbits = **ceiling**($\log_2$(b-a))-**floor**($\log_2$(g))+1
   = **ceiling**($\log_2$(19,000-(-900)-**floor**($\log_2$(0.5))+1 = 17
2. L = **ceiling**(Lbits/8) = **ceiling**(17/8) = 3
3. Follow steps in Starting Point B

Starting Point B - Algorithm: Compute sF, sR and Z_{offset}

1. bPow = **ceiling**($\log_2$(b-a)) = **ceiling**($\log_2$(19,000-(-900))) = 15
2. dPow = 8*L-1 = 8*3-1 = 23
3. sF = 2^(dPow -bPow) = 2^(23-15) = 2^8
4. sR = 2^(bPow- dPow) = 2^(15-23) = 2^(-8)
5. Z_{offset} = 0.0
6. if (a<0 and b>0) then Z_{offset} = sF*a-**floor**(sF*a)
   = (2^8)*(-900)-**floor**((2^8)*(-900)) = 0.0

Forward Mapping: Let x = 10.0 (underlined section is the algorithm exercised)

1. If x is a special value, then:
   y = Table Lookup to map to specified bit pattern
2. Else x is a normal floating-point number then:
   y = **truncate**(sF*(x-a)+Z_{offset}) = **truncate**(2^8*(10.0-(-900))+0.0) = 232,960
   = 0x03 8E00

Reverse Mapping: Let y = 232,960 = 0x03 8E00

Define: Bit(q,y) = the qth bit of y. msb = Most significant bit

1. special_value = Bit(msb,y) & Bit(msb-1,y)   = 0 & 0
2. If special_value equals 1 then:
   x = Table Lookup to map to specified floating-point value
3. Else y is a normal value then:
   x = sR*(y-Z_{offset})+a = (2^(-8)) * (232,960-0.0) + (-900) = 10.0

Table 6 shows some forward mapping and reverse mapping values for this example.

Table 6: Forward/Reverse mapping for Example 3

x	y	y (hex)	x mapped back
-900.0	0	0x00 0000	-900.0
0.0	230,400	0x03 8400	0.0
10.0	232,960	0x03 8E00	10.0
-infinity	See hex	0xE8 0000	-infinity

The following illustrates a complete example of the computations for mapping the small range using, IMAPB(0.1, 0.9, 2).

Example 4: Small range (Starting point B):

- a = Floating-Point Minimum = 0.1
- b = Floating-Point Maximum = 0.9
- L = 2 bytes

Starting Point B - Algorithm: Compute sF, sR and Z_{offset}

1. bPow = **ceiling**($\log_2$(b-a)) = **ceiling**($\log_2$(0.9-(0.1))) = 0
2. dPow = 8*L-1 = 8*2-1 = 15
3. sF = 2^(dPow -bPow) = 2^(15-0) = 2^15
4. sR = 2^(bPow- dPow) = 2^(0-15) = 2^(-15)
5. Z_{offset} = 0.0
6. if (a<0 and b>0) then Z_{offset} = sF*a-**floor**(sF*a)

Forward Mapping: Let x = 0.5 (underlined section is the algorithm exercised)

1. If x is a special value, then:
   y = Table Lookup to map to specified bit pattern
2. Else x is a normal floating-point number then:
   y = **truncate**(sF*(x-a)+Z_{offset}) = **truncate**(2^15*(0.5-(0.1))+0.0) = 13,107
   = 0x3333

Reverse Mapping: Let y = 13,107 = 0x3333

Define: Bit(q,y) = the qth bit of y. msb = Most significant bit

1. special_value = Bit(msb,y) & Bit(msb-1,y) = 0 & 0
2. If special_value equals 1 then:
   x = Table Lookup to map to specified floating-point value
3. Else y is a normal value then:
   x = sR*(y-Z_{offset})+a = (2^(-15)) * (13,107-0.0) + (0.1) = 0.499993896484375

Table 7 shows some forward mapping and reverse mapping values for this example.

Table 7: Forward/Reverse mapping for Example 4

x	y	y (hex)	x mapped back	Error
0.1	0	0x0000	0.1	0.0
0.5	13,107	0x3333	0.499993896484375	6.10e-6
0.9	26,214	0x6666	0.89998779296875	1.22e-5
-infinity	See hex	0xE800	-infinity	n/a